Prediction of Heterogeneous Device Task Runtime Based on Edge Server-Oriented Deep Neuro-Fuzzy System

Haijie Wu, Weiwei Lin, *Senior Member, IEEE*, Wangbo Shen, Xiumin Wang, C. L. Philip Chen, *Fellow, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—Predicting the runtime of tasks is of great significance as it can help users better understand the future runtime consumption of the tasks and make decisions for their heterogeneous devices, or be applied to task scheduling. Learning features from user task history data for predicting task runtime is a mainstream method. However, this method faces many challenges when applied to edge intelligence. In the Big Data era, user devices and data features are constantly evolving, necessitating frequent model retrains. Meanwhile, the noisy data from these devices requires robust methods for valuable insight extraction. In this paper, we propose an edge server-oriented deep neuro-fuzzy system (ESODNFS) that can be trained and inferred on edge servers, for providing users with task runtime prediction services. We divided the dataset and trained it on multiple improved adaptive-network-based fuzzy inference system units (ANFISU), and finally conducted joint training on a deep neural network (DNN). By partitioning the dataset, we reduced the number of parameters for each ANFISU, and at the same time, multiple units can be trained in parallel, supporting fast training and iteration. Additionally, the application of fuzzy inference can effectively learn the features in noisy data and make accurate predictions. The experimental results show that ESODNFS can accurately predict the runtime of real tasks. Compared with other DNN and DNFS, it can achieve good prediction results while reducing training time by over 35%.**

***Index Terms*—Deep neuro-fuzzy system, edge server, heterogeneous device, prediction.**

I. Introduction

As A popular direction of cloud computing, edge computing has been widely studied and applied in recent years. Edge computing solves the problem that the delay of data transmission to the cloud computing center is too long in the traditional cloud computing framework. By sinking computing and data processing capabilities to the edge of the network, edge servers can respond more quickly to user needs and provide real-time services. With the development of the Internet of Things and artificial intelligence technology, a new technology that combines edge computing and artificial intelligence, edge intelligence, has emerged. Edge intelligence deploys AI algorithms and models to edge servers, providing fast response while also providing intelligent services.

Edge intelligence has played a huge driving role in the development of the Internet of Things. It is the promotion and application of edge intelligence that enables industries such as autonomous driving and smart cities to develop. Therefore, in recent years, there have been many studies related to edge intelligence. The authors of [1] conducted a comprehensive analysis of current and emerging edge computing architectures for smart healthcare. They also examined the application of advanced artificial intelligence techniques for classification and prediction in edge intelligence. A self-learning architecture based on self-supervised generative adversarial nets was proposed and was proved to have the potential to identify and classify unknown services that emerge in edge computing networks [2]. A blockchain-based edge intelligence system was presented by [3] to ensure the CEDs' data security, privacy, latency, and efficiency and presented the use case scenario of blockchain and edge intelligence in the COVID-19 pandemic. In [4], the challenge of edge intelligence and tiny machine learning was assessed and the paper highlighted their issues with a particular focus on microcontroller deployment and offered potential solutions. These studies indicate that edge intelligence is becoming a research hotspot in the Internet of Things, driving the development of cloud computing and playing an increasingly important role in industrial development.

With the help of edge intelligence, many technologies and services that require quick response and AI algorithm support can be provided. For example, predicting the duration of task execution in the current device state based on the user's task running history. The runtime of a task often depends on many factors, such as device architecture, resources, network status, etc., and obtaining the runtime of a task in advance is very

Received 24 July 2024; revised 16 November 2024; accepted 13 December 2024. Date of publication 23 December 2024; date of current version 6 February 2025. This work was supported in part by National Natural Science Foundation of China under Grant 62072187, in part by the Major Key Project of PCL, China under Grant PCL2023A09, and in part by Guangzhou Development Zone Science and Technology Project under Grant 2023GH02. *(Corresponding author: Weiwei Lin.)*

Haijie Wu, Wangbo Shen, Xiumin Wang, and C. L. Philip Chen are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: 202030442496@mail.scut.edu.cn; 202010107337@mail.scut.edu.cn; xmwang@scut.edu.cn; philip.chen@ieee.org).

Weiwei Lin is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, and also with Pengcheng Laboratory, Shenzhen 518055, China (e-mail: linww@scut.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TSC.2024.3520869

meaningful. First, knowing the runtime of a task in advance is beneficial for users to make better decisions on whether to run the task or execute other alternative tasks. Second, terminal device resources are gradually becoming heterogeneous, and many devices have multiple types of heterogeneous resources (such as CPU, GPU, NPU) at the same time [5]. Tasks may have multiple ways of running, each using different resources and resulting in significant differences in runtime [6]. Third, many scheduling algorithms are designed to utilize the runtime of tasks for achieving the shortest completion time, both in cloud [7], [8], [9] and edge [10], [11] scenarios. However, inaccurate task runtime can lead to a decrease in the performance of scheduling algorithms, so accurate task runtime prediction is helpful for the design of scheduling algorithms.

The mainstream approach to predicting task runtime leverages machine learning techniques on historical data, aiming to discern patterns of runtime fluctuations from past records [12]. The main difficulties of this approach are twofold. On the one hand, it is important to select features that are closely related to the runtime of the task for prediction and are often related to the content of the task itself. On the other hand, there is often noise and missing data in historical data, which places great demands on the robustness of the model. Deep Neuro-Fuzzy System (DNFS) performs well on these issues due to its strong robustness, generalization, and interpretability [13].

DNFS is a technology that combines DNN and Fuzzy Logic (FL). DNN has been extensively studied and applied in many industries in recent years, such as transportation [14], medicine [15], biological recognition [16], and so on. It is precisely because of DNN's powerful learning and memory capabilities that it can complete many complex classification, prediction, and generation tasks. However, due to its black-box drawback, DNN has poor interpretability, making it difficult to understand the inference process of the network [17], and even bringing security risks [18]. FL has been applied and studied in many industries by using IF-THEN rules to simulate human reasoning processes, and has made great progress [19], [20], [21]. A technique of combining neural networks with FL called adaptive-network-based fuzzy inference system (ANFIS), was proposed [22] and has been proven to have good robustness and generalization in prediction [23]. More importantly, since ANFIS builds a network based on the process of FL, its network structure is interpretable, and the calculation and inference processes in the network can be explained using the theory of fuzzy inference. Therefore, DNFS combines FL and DNN, effectively integrating the learning ability of DNN with the interpretability of FL.

In recent years, there have been many studies and applications related to DNFS. In [24], a self-organizing DNFS was proposed to classify kidney cancer subgroups and it improved that DNFS can be very useful in high dimensional data. [25] integrated DNFS with a butterfly optimization algorithm to improve the prediction accuracy on forecasting the PM2.5 concentration. DNFS was used to calculate the risk levels for patients and provide patients with the potential recommendation about the severity staging of the associated diseases [26]. These studies indicate that DNFS has high research value and performs well in many applications. Therefore, DNFS will be able to perform well in predicting task runtime.

However, few works effectively combine DNFS and edge intelligence. In the Big Data era, the swift evolution of terminal device information and associated data features necessitates frequent model retraining using the most recent device data since the old model may suffer performance degradation in the presence of new data. This process often imposes a significantly long training model time on edge servers. Furthermore, the presence of noise and missing data in the information collected from terminal devices often hampers the performance of conventional deep learning models in extracting valuable insights. Hence, a DNFS that embodies robustness, accuracy, and swift training capabilities could effectively address these challenges. Regrettably, such works are still lacking.

Considering the aforementioned shortcomings, we propose ESODNFS for deployment on edge servers and receive task runtime prediction requests from heterogeneous end devices, providing accurate predictions. We summarize the contributions and innovations as follows:

- We introduce a DNFS-based model for predicting task runtime, called ESODNFS, which provides accurate predictions based on device resource conditions before task execution. This method compensates for the shortcomings of traditional task runtime prediction models in terms of robustness, generalization, accuracy, and interpretability.
- The ESODNFS we proposed can be trained in parallel. We divide the dataset and train each part with different ANFISUs. Then, we perform joint training using a DNN to weigh the output of each unit. Our proposed ANFISU is a model that optimizes the membership function (MF) and rules of classical ANFIS, with more flexible parameter settings and better generalization ability. We used least squares estimation (LSE) and particle swarm optimization (PSO) instead of gradient descent to achieve better training efficiency. We also use PSO to learn the parameters of DNN to achieve better model accuracy. This method can reduce the number of parameters in DNFS and improve the training effect of the model, while also completing fast training in parallel.
- We collect task data from real heterogeneous devices and implement and train the proposed model, verifying the effectiveness and feasibility of our method through comparative experiments.

The rest of this paper is organized as follows: Section II introduces some related work. Sections III and IV introduce the proposed models and methods. Section V conducts experiments and analyses of experimental results. Section VI concludes the paper with future research directions.

II. Background and Related Work

A. Methods for Predicting Task Runtime

There are many research methods for predicting task runtime, and the methods used vary according to the different characteristics of the task. In [27], the authors highlighted the significance

of predicting task runtime for Big Data platforms like MapReduce, Hadoop, and Spark, which is crucial for scheduling, cost estimation, and cluster management. They proposed a gray box modeling approach for runtime prediction on Spark. A method that improves task runtime predictions for high-performance scheduling was developed by [28]. This method clusters historical task logs to categorize tasks and selects six features with strong runtime correlations. It uses a Transformer model with plain connections and an attention mechanism, resulting in enhanced prediction accuracy. The paper [29] also proposes a method to improve scheduling performance by introducing multiple additional task features to determine the task execution mode and obtain a refined model. Through two-step task runtime estimation, more accurate predictions are obtained. The authors in [30] argue that scheduling optimization of jobs requires knowledge of the runtime estimation of running jobs and jobs in queues. When the scheduler lacks techniques to compute job runtimes, users are asked to provide runtimes for submitted jobs, which are often inaccurate. Therefore, a method based on machine learning and genetic algorithms has been proposed for predicting the runtime of HPC jobs, thus helping users to evaluate job runtimes more accurately. A similar idea was proposed by [31], where a simple classifier is used to estimate job runtimes and categorize jobs into large and small. A novel running time prediction framework, PREP was proposed for achieving more accurate predictions [32]. PREP takes the running path of a job as a new feature and clusters the running path based on the K-means algorithm. Eventually, some regression models in machine learning are used to achieve the prediction. These studies further demonstrate the importance of predicting task runtime and provide some ideas for prediction.

B. Optimization Methods of DNFS and Their Application in Prediction

Essentially, DNFS is a combination of FL and DNN, hence many optimizations applicable to FL and DNN can also be utilized for DNFS. The method of non-gradient updating network parameters, such as metaheuristic algorithms, is widely used in ANFIS and has excellent performance. The authors of [33] proposed a three-stage fuzzy metaheuristic algorithm (TSFM) to implement an adaptive charging scheduling algorithm to meet wireless rechargeable sensor networks with multi-objective requirements. Whale Optimization Algorithm (WOA) is applied to parameterize fuzzy rules to enhance the algorithm's performance. The paper [34] introduced a fuzzy system incorporating Grey Wolf Optimizer (GWO) for solving the routing problem in wireless body area networks. GWO adjusts the hyperparameters in the fuzzy system including the parameters of the rules. In [35], FL control is applied to the energy management system's design of an isolated microgrid, and the parameters of FL control are optimized by PSO and Cuckoo Search.

In terms of the combination of FL and DNN, it can be roughly divided into three types [13], [17]. Sequential DNFS first fuzzifies the input, then learns through DNN, and finally outputs the prediction results. Parallel DNFS performs both fuzzification and DNN learning on the input, combines the results, and outputs them. Collaborative DNFS is based on sequential DNFS, where the output of sequential DNFS is further defuzzifying as the final output. Due to the structural linearity of sequential DNFS, it is considered a slow model, while the other two structures have more flexibility in learning and interpretability when dealing with complex real-world problems of large-scale data [13]. Therefore, in future research, there may be a greater emphasis on the design of parallel DNFS and cooperative DNFS.

Many studies have utilized DNFS for various predictive applications. A deep spatiotemporal fuzzy neural network was proposed for predicting subway passenger flow during COVID-19 [36]. The authors point out that the difficulty in predicting subway passenger flow lies in the complex and nonlinear influencing factors of pedestrian flow. However, the introduction of fuzzy neural networks has effectively solved this problem. The paper [37] utilizes DNFS to predict the optimal worker task matching to address the bilateral location privacy-preserving problem in mobile crowd sensing scenarios. DNFS shows good prediction performance in obfuscated location information. In [38], a novel deep fuzzy network was developed for RNA N6-Methyladenosine sites prediction and it shows that the application of fuzzy systems to deep learning can improve the generalizability and robustness of models. A hierarchical Pythagorean fuzzy deep neural network was proposed by [39] for predicting the quality requirements of cloud services. It adopted the structure of parallel DNFS and enhanced the model learning ability using DNN at the end, achieving more accurate prediction results compared to ordinary DNN and fuzzy neural networks. These successful applications in prediction problems demonstrate the effectiveness of DNFS in prediction, however, most of these research works do not consider applications in edge intelligence, nor have they been applied to predict task runtime.

III. Hyperparameter Setting and Training Method of ANFISU

ANFISU is one of the components of ESODNFS proposed, which is an improvement on ordinary ANFIS aimed at improving model performance and reducing computational power requirements, making it suitable for deployment on edge servers. Therefore, in this section, we first briefly describe the architecture of ANFIS, and then present the design of the MFs, the determination of the hyperparameters, and the training method in the proposed ANFISU.

A. ANFIS Architecture

ANFIS is a neural network structure based on the Takagi-Sugeno (T-S) fuzzy model. In the T-S fuzzy model, the consequent of the rule is the output of a linear function. The form of rules is defined as follows:

$$R^{(k)}: \text{ If } x_1 \text{ is } A_1^{(k)} \text{ and } x_2 \text{ is } A_2^{(k)} \text{ and } \ldots \text{ and } x_d \text{ is } A_d^{(k)},$$
$$\text{Then } y_k = p_0^{(k)} + \sum_{i=1}^{d} p_i^{(k)} x_i \tag{1}$$

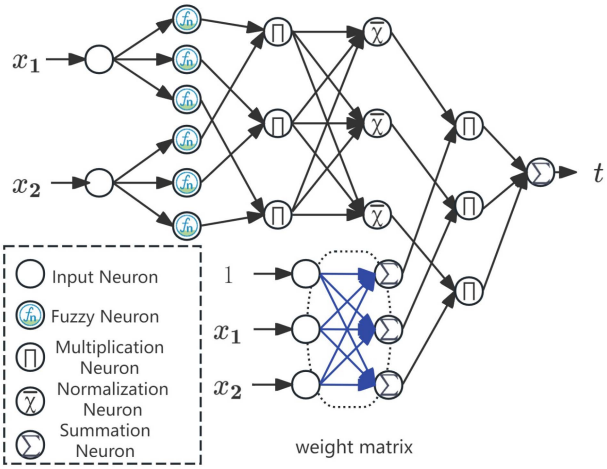

Fig. 1. Schematic diagram of ANFIS structure, where Fuzzy Neuron represents the calculation of MF. The forward propagation process of each input is essentially the calculation process of FL.

where $R^{(k)}$ indicates the kth rule, $k = 1, 2, \ldots, K$ and K is the number of rules, x_i is the ith dimension of input and $A_i^{(k)}$ indicates the corresponding fuzzy set and $i = 1, 2, \ldots, d$, d indicates the dimension number of input, y_k indicates the output of the kth rule, $p_i^{(k)}$ indicates the weight of ith dimension of input in the kth rule. In the process of inference, first, for an input $\boldsymbol{X} = (x_1, x_2, \ldots, x_d)$, we calculate the membership degrees of all dimensions for the kth rule as $(A_1^{(k)}(x_1), A_2^{(k)}(x_2), \ldots, A_d^{(k)}(x_d))$, where $A_i^{(k)}(\cdot)$ indicates the MF of the ith dimension of input in the kth rule. Therefore, the firing strength of each rule is calculated by the following equation:

$$f_k(\boldsymbol{X}) = \prod_{i=1}^{d} A_i^{(k)}(x_i) \tag{2}$$

where $f_k(\boldsymbol{X})$ indicates the firing strength of the kth rule of the input $\boldsymbol{X}$, $k = 1, 2, \ldots, K$. The firing strengths need to be normalized, denoted as $(\overline{f_1(\boldsymbol{X})}, \overline{f_2(\boldsymbol{X})}, \ldots, \overline{f_K(\boldsymbol{X})})$, where

$$\overline{f_k(\boldsymbol{X})} = \frac{f_k(\boldsymbol{X})}{\sum_{i=1}^{K} f_i(\boldsymbol{X})}, \quad k = 1, 2, \ldots, K \tag{3}$$

Meanwhile, for a known rule $R^{(k)}$, $p_0^{(k)}, p_1^{(k)}, \ldots, p_d^{(k)}$ are known values, so the output for all rules can be calculated as $(y_1, y_2, \ldots, y_K)$. The final output equation is:

$$\hat{t} = \sum_{i=1}^{K} \overline{f_i(\boldsymbol{X})} y_i \tag{4}$$

where $\hat{t}$ indicates the inference result of the input $\boldsymbol{X}$. This process is expressed using a neural network, as shown in Fig. 1, and this network is ANFIS.

The only unknowns in ANFIS are the weights of rules and the parameters in the MFs. The number and type of parameters for MFs vary depending on the choice of MF. These parameters are obtained through learning on the dataset, which is essentially the learning of rules. This means that for a trained ANFIS, it can be explained through a T-S fuzzy model.

B. Design of MFs

In many ANFIS studies, Gaussian MF is used as the selection of MF due to its many good properties such as differentiability and symmetry. In [39], Gaussian MF was used, and non-membership degree was introduced based on the Pythagorean fuzzy set to enhance the expression of MF. In the design of ANFISU, we also use Gaussian MF and Pythagorean fuzzy sets. We put the computation process of membership degree and non-membership degree in fuzzy neuron (FN), which we show in Fig. 1 and represents the results of MF computation in ANFIS. They are calculated by the following equations:

$$\mu_i^{(k)} = \exp\left(-\frac{(x_i - c_{i,1}^{(k)})^2}{2(\sigma_{i,1}^{(k)})^2}\right) \tag{5}$$

$$\nu_i^{(k)} = \exp\left(-\frac{(x_i - c_{i,2}^{(k)})^2}{2(\sigma_{i,2}^{(k)})^2}\right) \tag{6}$$

where $\mu_i^{(k)}$ and $\nu_i^{(k)}$ indicate the membership degree and non-membership degree of the kth rule of the ith dimension of input respectively, $0 < \mu_i^{(k)} < 1$, $0 < \nu_i^{(k)} < 1$. $c_{i,1}^{(k)}$, $c_{i,2}^{(k)}$ and $\sigma_{i,1}^{(k)}$, $\sigma_{i,2}^{(k)}$ are the means and standard deviations, respectively, which are determined by training. Gaussian MF and Pythagorean fuzzy sets give ANFISU a good fuzzy representation. With the use of non-membership degree, the Pythagorean fuzzy set can characterize uncertainty more accurately and have a more powerful representation, while also bringing in more parameters. However, the Gaussian MF has few parameters and provides smoothness and interpretability. Therefore, this combination enables more accurate fuzzy inference with as few parameters as possible. In the Pythagorean fuzzy set, the membership degree and non-membership degree have the following constraint:

$$0 \le \left(\mu_i^{(k)}\right)^2 + \left(\nu_i^{(k)}\right)^2 \le 1 \tag{7}$$

However, the constraint is difficult to maintain during the parameter learning process of FNs. When the parameters of the FNs violate the constraint during learning, some of the parameters need to be tuned to satisfy the constraint, thus increasing the difficulty of training and potential performance degradation. Moreover, considering which parameters need to be tuned makes the model more complex. Therefore, instead of tuning the original parameters of FNs to satisfy the constraint, we introduce the bound index λ for optimization. Formally, the constraint for μ and ν is as follows:

$$0 \le \left(\mu_i^{(k)}\right)^{\lambda_i^{(k)}} + \left(\nu_i^{(k)}\right)^{\lambda_i^{(k)}} \le 1 \tag{8}$$

where $\lambda_i^{(k)}$ indicates the bound index of the kth rule of the ith dimension of input. The bound index provides dynamic constraints for membership degree and non-membership degree and scales them at the FN output. This scaling is achieved by using the bound index as an exponent, thus the scaled membership degree and non-membership degree remain in the interval (0,1). To get a valid bound index to satisfy the constraint quickly, the bound index needs to be computed explicitly. However, the range of

the bound index cannot be solved exactly by (8), so we perform the following transformation:

$$\left(\mu_i^{(k)}\right)^{\lambda_i^{(k)}} + \left(\nu_i^{(k)}\right)^{\lambda_i^{(k)}} \leq 2\max\left(\mu_i^{(k)}, \nu_i^{(k)}\right)^{\lambda_i^{(k)}} \leq 1 \quad (9)$$

Thus, it can be inferred that:

$$\lambda_i^{(k)} \geq \frac{\ln 2}{\min\left(\frac{(x_i - c_{i,1}^{(k)})^2}{2(\sigma_{i,1}^{(k)})^2}, \frac{(x_i - c_{i,2}^{(k)})^2}{2(\sigma_{i,2}^{(k)})^2}\right)} \quad (10)$$

This indicates that the constraints of (8) will always hold as long as μ and ν satisfy the above inequality. In order for the inequality to hold, $\lambda_i^{(k)}$ will not be learned during the ANFISU training process but will be randomly selected each time within the appropriate range. We define

$$(\lambda_{\min})_i^{(k)} = \frac{\ln 2}{\min\left(\frac{(x_i - c_{i,1}^{(k)})^2}{2(\sigma_{i,1}^{(k)})^2}, \frac{(x_i - c_{i,2}^{(k)})^2}{2(\sigma_{i,2}^{(k)})^2}\right)} \quad (11)$$

Due to $0 < \mu_i^{(k)} < 1$, $0 < \nu_i^{(k)} < 1$, an excessively large $\lambda_i^{(k)}$ will result in both $(\mu_i^{(k)})^{\lambda_i^{(k)}}$ and $(\nu_i^{(k)})^{\lambda_i^{(k)}}$ becoming very small, thus the upper bound $(\lambda_{\max})_i^{(k)}$ needs to be determined when $\lambda_i^{(k)}$ is selected randomly, and we set $(\lambda_{\max})_i^{(k)} = 2$ in this paper. $\lambda_i^{(k)}$ is specified by the following equation:

$$\lambda_i^{(k)} = \begin{cases} (\lambda_{\min})_i^{(k)}, & (\lambda_{\min})_i^{(k)} > (\lambda_{\max})_i^{(k)} \\ \lambda_i^{(k)} \sim U[(\lambda_{\min})_i^{(k)}, (\lambda_{\max})_i^{(k)}], & \text{otherwise} \end{cases} \quad (12)$$

where $U[-a, a]$ is the uniform distribution within the range of $(-a, a)$. Finally, the output of FN is expressed as:

$$B_i^{(k)}(x_i) = \max\left(0, \left(\mu_i^{(k)}\right)^{\lambda_i^{(k)}} - \left(\nu_i^{(k)}\right)^{\lambda_i^{(k)}}\right) \quad (13)$$

where $B(\cdot)$ represents the output of FN, and $A(\cdot)$ in (2) needs to be replaced with $B(\cdot)$. The randomized bound index avoids the increase of the parameters to be trained, thus constraining the unreasonable membership degree and non-membership degree without increasing the training complexity. Meanwhile, the randomness of the network will be able to enhance the generalization of the model and prevent overfitting.

C. Determination of the Number of FNs and Rules

In the original ANFIS design, each input dimension is fuzzified through K MFs, which are combined to form K rules without repetition. This structure brings about inflexibility in the model, and due to the independent relationships between rules, it cannot express complex reasoning in reality. Moreover, this design will bring a large number of parameters to be optimized. In our previous modeling, each FN had 5 unknowns. If the input dimension is 4 and the number of rules is 10, there are a total of $4 \times 10 \times 5 = 200$ parameters in all FNs. Due to our ESODNFS being aimed at edge servers, having too many parameters brings a significant computational burden. Therefore, we have designed a method for determining the number of FNs and a method for designing rules.

First, we define the number of FNs for each input dimension as $(a_1, a_2, \ldots, a_d)$. Intuitively, more FNs should be used for fuzzification in dimensions that are more conducive to prediction. Our method is to define the upper bound $a_{\max}$ and lower bound $a_{\min}$, then calculate the value of each dimension and determine the number of FNs between $a_{\max}$ and $a_{\min}$ based on the ranking of values. For the value of each dimension, we use entropy to define it, which describes the distribution of data in that dimension in the dataset. The more balanced the distribution of data, the greater the entropy, and the stronger the decisiveness of that dimension in prediction. We define the entropy $ent(\boldsymbol{b})$ of an array $\boldsymbol{b} = \{b_1, b_2, \ldots, b_{|\boldsymbol{b}|}\}$ to be calculated using the following method. First, define $b_{min}, b_{\max}$ are the minimum and maximum values of array $\boldsymbol{b}$, and $\boldsymbol{b'} = \{b'_1, b'_2, \ldots, b'_{|\boldsymbol{b'}|}\}$ is obtained from $\boldsymbol{b}$, where

$$b'_i = \frac{b_i - b_{\min}}{b_{\max} - b_{\min}}, \quad i = 1, 2, .., |\boldsymbol{b}| \quad (14)$$

Furthermore, the array $\boldsymbol{b''} = \{b''_1, b''_2, \ldots, b''_{|\boldsymbol{b''}|}\}$ is obtained where

$$b''_i = \frac{b'_i}{\sum_{j=1}^{|\boldsymbol{b'}|} b'_j}, \quad i = 1, 2 \ldots, |\boldsymbol{b'}| \quad (15)$$

Finally,

$$ent(\boldsymbol{b}) = \sum_{i=1}^{|\boldsymbol{b''}|} b''_i \log_2(b''_i) \quad (16)$$

where $b''_i \log_2(b''_i) = 0$ if $b''_i = 0$. When all values in array $\boldsymbol{b}$ are the same, set $ent(\boldsymbol{b}) = 0$ without going through the above calculation process.

Therefore, we divide all samples in the dataset into d arrays based on their dimensions and use $ent(\cdot)$ to calculate d arrays to obtain $\{e_1, e_2, \ldots, e_d\}$ d values. The number of FNs for each dimension is determined by the following equation:

$$a_i = a_{\min} + \left\lfloor \frac{(a_{\max} - a_{\min}) \sum_{j=1}^{d} [e_i > e_j]}{d} \right\rfloor \quad (17)$$

where $i = 1, 2, \ldots, d$, $[e_i > e_j] = 1$ if $e_i > e_j$ is true and $[e_i > e_j] = 0$ if $e_i \leq e_j$. In this way, more FNs will be available for dimensions that have a more even distribution of data and are more decision-making.

After determining the number of FNs for each dimension, we consider combining rules from these FNs. The maximum number of rules is $\prod_{i=1}^{d} a_i$, which is huge when there are many FNs and dimensions. Therefore, our method is to select the best K rules from them. For ease of expression, we define a rule in the form of $R = (r_1, r_2, \ldots, r_d)$, where r_i indicates that this rule uses the r_ith FN of the ith dimension, where $i = 1, 2, \ldots, d$. We assume $d = 3$, the number of FNs is (3, 2, 2), and set the number of rules $K = 3$. As shown in Fig. 2, the three rules in Fig. 2(a) are represented as (1, 1, 1), (2, 2, 2), and (3, 1, 2), respectively and the three rules in Fig. 2(b) are (1, 1, 1), (1, 1, 2), and (1, 2, 1). Obviously, the similarity between the rules in Fig. 2(b) is high, and the FNs have not been fully utilized. For two similar rules,

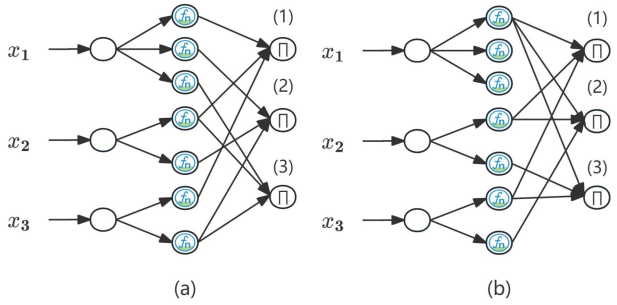

Fig. 2. Schematic diagram of rule combination in cases where the number of membership functions in each dimension is different. For example, in (a), rule (3) consists of the 3rd FN of the 1st dimension, the 1st FN of the 2nd dimension, and the 2nd FN of the 3rd dimension, and is thus denoted as (3, 1, 2).

e.g., (1, 1, 1) and (1, 1, 2), they use two identical FNs, so the difference between the information contained in these two rules lies mainly in the third dimension of the data, and the differences in the other dimensions of the data are not well reflected in the output of the rules. To make the rule set more valuable, the FNs contained in these rules need to be as different as possible. On the one hand, the rules in the rule set need to maximize their differences, which means that two rules duplicate the least amount of FNs. On the other hand, all FNs need to be utilized evenly. Therefore we design the K-rule algorithm.

We define S as the set of selected rules, and T as the set of remaining rules, and the total size of the two sets is $\prod_{i=1}^{d} a_i$. Initially, $S = \varnothing$ and $|T| = \prod_{i=1}^{d} a_i$. We define the distance $dis(R, S)$ from rule R to set S as determined by the following equation:

$$dis(R, S) = \min_{R' \in S} \varphi(R, R') + g(R, S) \tag{18}$$

Among them, $\varphi(R, R')$ represents the difference between two rules, reflecting the degree of association of the rules,

$$\varphi(R, R') = \sum_{i=1}^{d} [R_i \neq R'_i] \tag{19}$$

$g(R, S)$ reflects the rarity of each dimension of the rule in set S,

$$g(R, S) = \sum_{i=1}^{d} \frac{\sum_{R' \in S} [R_i \neq R'_i]}{|S|} \tag{20}$$

Therefore, we calculate the dis values of all rules in set T, take the rule s with the largest dis, and then $S \cup \{s\}, T \setminus \{s\}$. Repeating K times yields the K rules, and these rules have the greatest difference, making better use of the information in FNs. It should be noted that different rules may use the same FN multiple times, for example, the first FN in the second dimension of Fig. 2(a) is used by the first and third rules, so $B_2^{(1)}$ and $B_2^{(3)}$ are the same FN.

D. ANFISU Training Method

In the ANFISU we designed, the parameters to be learned consist of the parameters in the FNs and the weights of the rules. Metaheuristic algorithms are considered to be a more efficient method compared to gradient descent for training the parameters of ANFIS [17]. Since the output of each rule is calculated through a linear function, LSE can quickly calculate the optimal weights of rules without iteration. Therefore, we use PSO to learn the FN parameters and calculate the weights of rules using LSE each time, with the minimum mean-square error (MMSE) as the objective function for optimization.

Specifically, the encoding of PSO is to sequentially form a position vector $\boldsymbol{pos}$ with a dimension of $4\sum_{i=1}^{d} a_i$ from FN parameters, and update it in each round. We define the two acceleration coefficients as c_1 and c_2 respectively, and the inertia weight is defined as w. We use a linearly decreasing inertia weight, where the maximum and minimum values are $w_{\max}$ and $w_{\min}$. A $\boldsymbol{pos}$ determines all FNs, so for an input $\boldsymbol{X}$, $(\overline{f_1(\boldsymbol{X})}, \overline{f_2(\boldsymbol{X})}, \ldots, \overline{f_K(\boldsymbol{X})})$ can be calculated. According to (4), the following equation can be further calculated:

$$\begin{aligned}\hat{t} &= \sum_{i=1}^{K} \overline{f_i(\boldsymbol{X})} \left(p_0^{(i)} + \sum_{j=1}^{d} p_j^{(i)} x_j \right) \\ &= \overline{f_1(\boldsymbol{X})} p_0^{(1)} + \overline{f_1(\boldsymbol{X})} x_1 p_1^{(1)} + \cdots + \overline{f_1(\boldsymbol{X})} x_d p_d^{(1)} \\ &\quad + \overline{f_2(\boldsymbol{X})} p_0^{(2)} + \overline{f_2(\boldsymbol{X})} x_1 p_1^{(2)} + \cdots + \overline{f_K(\boldsymbol{X})} x_d p_d^{(K)}\end{aligned} \tag{21}$$

We define $\boldsymbol{P} = (p_0^{(1)}, p_1^{(1)}, \ldots, p_d^{(1)}, p_0^{(2)}, p_1^{(2)}, \ldots, p_d^{(K)})^T$ which size is $(K(d+1) \times 1)$ and assume that there are a total of N samples in the dataset, defined as $\boldsymbol{X'}$, with each sample defined as $\boldsymbol{X'_i} = (x_{i,1}, x_{i,2}, \ldots, x_{i,d})$, $i = 1, 2, \ldots, N$. The matrix $\boldsymbol{F}$ is defined as follows:

$$\boldsymbol{F} = \begin{Bmatrix} \overline{f_1(\boldsymbol{X'_1})} & \overline{f_1(\boldsymbol{X'_1})} x_{1,1} & \cdots & \overline{f_2(\boldsymbol{X'_1})} & \cdots & \overline{f_K(\boldsymbol{X'_1})} x_{1,d} \\ \overline{f_1(\boldsymbol{X'_2})} & \overline{f_1(\boldsymbol{X'_2})} x_{2,1} & \cdots & \overline{f_2(\boldsymbol{X'_2})} & \cdots & \overline{f_K(\boldsymbol{X'_2})} x_{2,d} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \overline{f_1(\boldsymbol{X'_N})} & \overline{f_1(\boldsymbol{X'_N})} x_{N,1} & \cdots & \overline{f_2(\boldsymbol{X'_N})} & \cdots & \overline{f_K(\boldsymbol{X'_N})} x_{N,d} \end{Bmatrix} \tag{22}$$

with a size of $(N \times K(d+1))$. The label of the dataset is $\boldsymbol{Y}$ which size is $(N \times 1)$, thus the loss function can be expressed as:

$$\mathcal{L}(\boldsymbol{P}) = (\boldsymbol{FP} - \boldsymbol{Y})^T (\boldsymbol{FP} - \boldsymbol{Y}) \tag{23}$$

According to LSE, the optimal $\boldsymbol{P}$ that can minimize the loss function can be calculated using the following equation:

$$\boldsymbol{P} = (\boldsymbol{F^T F})^{-1} \boldsymbol{F^T Y} \tag{24}$$

Therefore, for the encoding of a particle swarm, we can calculate the weights of rules using (24) to determine all parameters of the model and calculate the predicted values of the model. At the same time, mean-square error (MSE) serves as the fitness function. It is worth noting that $\boldsymbol{F^T F}$ in (24) is not always an invertible matrix, so singular value decomposition (SVD) is needed to solve the pseudo-inverse matrix of $\boldsymbol{F}$ and replace $(\boldsymbol{F^T F})^{-1} \boldsymbol{F^T}$ to ensure that $\boldsymbol{P}$ is always available.

Algorithm 1: Structure Determination and Training of ANFISU.

Data: $\boldsymbol{X'}, \boldsymbol{Y}$.
Result: A trained ANFISU.
Calculate the number of FNs for each dimension based on Eqs. (14), (15), (16), (17);
Calculate K rules based on Eqs. (18), (19), (20);
Initialize the positions and velocities of all particles;
for $epoch = 1$ *to* $epoch\_num$ **do**
  for $i = 1$ *to number of particles* **do**
    Update $\boldsymbol{pos}$ of the ith particle;
    Update the parameters of all FNs in ANFISU using $\boldsymbol{pos}$ of the ith particle;
    Calculate $\boldsymbol{F}$ and using LSE to calculate $\boldsymbol{P}$;
    Calculate MSE and update the optimal $\boldsymbol{pos}$;
Update ANFISU using the optimal $\boldsymbol{pos}$;
return ANFISU

At this point, the design of ANFISU has been fully introduced, and the pseudocode for the entire process is shown in Algorithm 1.

IV. Design of ESODNFS

The ANFISU we proposed already has predictive capabilities, but using it directly for task runtime prediction will lead to potential problems. ANFISU has a simple structure, insufficient depth, and limited learning ability. When the dataset size is large, the model's parameters, such as the number of FNs and rules, need to be correspondingly large. However, the model training process involves complex operations such as matrix multiplication and matrix inversion. For example, the matrix $\boldsymbol{F^T F}$ has size $(K(d+1) \times K(d+1))$, so the complexity of its inverse matrix computation for rule numbers is $\mathcal{O}(K^3)$. Moreover, such operations require each particle to run once in each round, resulting in excessive computational complexity. Therefore, we need to reduce the number of parameters in ANFISU. However, due to the decrease in the number of parameters, the learning ability of ANFISU will decrease, and it cannot learn the features in the dataset well, which can lead to underfitting problems.

Our approach is to evenly divide the dataset into s parts and construct s ANFISU models to learn different parts of the dataset separately. When making predictions, we input data onto each ANFISU and use a DNN model to weight s results. All ANFISUs and the DNN together form our proposed ESODNFS, as shown in Fig. 3. Due to the spatial parallelism of ANFISU and DNN, our ESODNFS can be regarded as a parallel DNFS. This design has the following advantages: 1) The size of the dataset learned by each ANFISU is reduced, and therefore the parameter size of the ANFISU is reduced accordingly. Since the training computational complexity increases cubically with the parameter size, even if multiple ANFISUs need to be trained, the overall training time is shorter than training one large-scale ANFISU. 2) DNN has stronger learning ability, however, traditional DNFS combines the structures of DNN and fuzzy neural networks, resulting in a decrease in interpretability. Our DNN

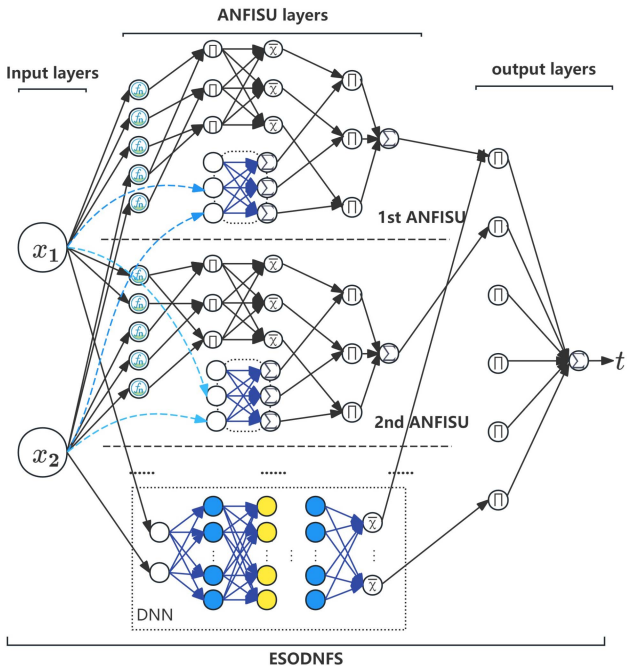

Fig. 3. ESODNFS consists of multiple ANFISUs and a DNN, with each input passing through all ANFISUs and DNNs to obtain multiple results in ANFISU and the weights of these results obtained in DNN.

is used for weighting multiple ANFISUs, enhancing the model learning ability without reducing the interpretability of DNFS. 3) These ANFISUs are independent and do not affect each other in training. Therefore, ESODNFS is able to train multiple ANFISUs simultaneously through parallel or distributed training instead of sequentially training these ANFISUs, thus realizing faster training speeds on edge servers. For example, to train ESODNFS on an edge server, the server creates s processes for training s ANFISUs simultaneously, fully utilizing multiple CPU resources. This parallelism speeds up the training of ANFISUs, thus reducing the overall training time of ESODNFS.

First, we introduce the designed DNN. Due to the wide range of applications of DNNs in real-world problems and the fact that we are using a common multilayer perceptron, we will not describe the specific structure. However, the following three points need to be noted:

Activation Function. We use the sigmoid activation function, and the equation is as follows:

$$\boldsymbol{o}^{(l)} = \frac{1}{1 + e^{-\left(\boldsymbol{w}^{(l)} o^{(l-1)} + \boldsymbol{b}^{(l)}\right)}} \tag{25}$$

where $\boldsymbol{o}^{(l)}$ indicates the output of the ith layer, $\boldsymbol{w}^{(l)}$ and $\boldsymbol{b}^{(l)}$ are the weight matrix and bias that connect the lth and $(l-1)$th layers, respectively.

Output. Since the output of DNN represents the weights of each ANFISU output, normalization is required in the final layer.

Training Method. Traditional DNNs use backpropagation to update parameters and accomplish training. Compared to the gradient-based training method, metaheuristic algorithms can overcome the defect of easily falling into local optima and be less likely to rely on the initial solution [40]. Due to the

Algorithm 2: Construction and Training of ESODNFS.

Data: $\boldsymbol{X'}, \boldsymbol{Y}$.
Result: A trained ESODNFS.
Divide the dataset $\boldsymbol{X'}$ and $\boldsymbol{Y}$ into s parts evenly;
Determine the hyperparameters of s ANFISUs and have each model learn one from s datasets based on Algorithm 1, getting s trained ANFISU;
Determine the number of layers and neurons in DNN;
Initialize the positions and velocities of all particles, whose encoding represents all weights of DNN;
for $epoch = 1$ *to* $epoch\_num$ **do**
  for $i = 1$ *to number of particles* **do**
    Update $\boldsymbol{pos}$ of the ith particle;
    Update the parameters of DNN using $\boldsymbol{pos}$ of the ith particle;
    Input $\boldsymbol{X'}$ into each ANFISU to obtain s prediction results;
    Input $\boldsymbol{X'}$ into DNN to obtain s weight;
    Use Eq. (26) to obtain the predicted value;
    Calculate MSE and update the optimal $\boldsymbol{pos}$;
Update DNN using the optimal $\boldsymbol{pos}$;
return ESODNFS

excessive computational burden, metaheuristic algorithms cannot be applied in large-scale DNNs. However, for ESODNFS, the size of DNN is designed to be small for faster training, thus DNN training based on metaheuristic algorithms will not impose a large computational burden and will be able to obtain better accuracy. Therefore, we use PSO for parameter learning. Specifically, all weights of each layer in DNN are sequentially encoded into $\boldsymbol{pos}$ and then optimized. Since all ANFISUs have been trained, samples are first input into each ANFISU to obtain s results. At the same time, samples are input into DNN to calculate s weights and weighted to obtain the output. This process is represented by the following equation:

$$\hat{t} = \sum_{i=1}^{s} DNN(\boldsymbol{X})_i \cdot ANFISU_i(\boldsymbol{X}) \tag{26}$$

where $\boldsymbol{X}$ represents a single input sample, $DNN(\cdot)_i$ represents the ith value output by DNN, $ANFISU_i(\cdot)$ represents the output of the ith ANFISU. MSE is also used as a fitness function. The training process of ESODNFS can be found in Algorithm 2.

Finally, we calculate the computational complexity of ESODNFS. For each ANFISU, the complexity of determining FNs and rules is $\mathcal{O}(\frac{N}{s}d + d^2)$ and $\mathcal{O}(\prod_{i=1}^{d} a_i K d)$, respectively. To avoid the generation of rules becoming a bottleneck, a subset of the rules in T are selected whose number is related to K, and hence the complexity is $\mathcal{O}(\frac{N}{s}d + K^2 d)$ with $K^2 d > d^2$. The computational complexity of LSE is $\mathcal{O}(\frac{N}{s}(\sum_{i=1}^{d} a_i + K^2\, d^2))$. Assume that the number of iteration rounds of PSO is n_1 and the number of particles is n_2, thus the computational complexity of the whole ANFISU is $\mathcal{O}(\frac{N}{s}d + K^2 d + \frac{n_1 n_2 N}{s}(\sum_{i=1}^{d} a_i + K^2\, d^2))$. The training complexity of the DNN is $\mathcal{O}(n_1 n_2 N(\sum_{i=1}^{d} a_i + Kd + W^2))$, where W is the size of the fully connected layer. Ultimately, the computational complexity of the entire ESODNFS is $\mathcal{O}(Nd + K^2 ds + n_1 n_2 N(\sum_{i=1}^{d} a_i + K^2\, d^2 + W^2))$. Compared to ANFIS, ESODNFS introduces more parameters, but the prediction accuracy and training time of ESODNFS are both superior to ANFIS, which will be verified in Section V.

V. Experiment

In this section, we conduct performance tests and experiments comparing the proposed ESODNFS with other methods. As ESODNFS is used to predict the runtime of end device tasks, experiments will also use running data of tasks from real heterogeneous devices for prediction.

A. Collection of Datasets

We have prepared three different types of tasks, namely AI-intensive, memory-intensive, and disk-intensive. For AI-intensive tasks, they can run with different resources based on the computing resources of the terminal devices, such as using GPU on devices with GPU resources and NPU on devices with NPU resources. For memory-intensive and disk-intensive tasks, they can only run using CPU resources.

On the end device, we use multiple Raspberry Pi, Atlas 200 DK, and PC devices as the end devices, where the Atlas 200 DK is equipped with NPU hardware resources. For the edge devices used for training and inference models, we use a device equipped with 8 Intel (R) Xeon (R) E5-2620 v4@2.10 GHz CPU(64 G) and with 2 Nvidia Tesla T4 GPU (16 G). Our dataset is obtained by running three different types of tasks multiple times on these devices. We have prepared two datasets with different dimensions and quantities. The first dataset only includes Raspberry Pi and PC devices running memory-intensive and disk-intensive task records, where device ID, task ID, CPU usage, CPU memory usage, disk usage, and task runtime are recorded as a sample of all information, totaling 6 dimensions, denoted as D1. The second dataset uses information from all devices running three types of tasks, with each sample adding GPU usage, GPU memory usage, NPU usage, and NPU memory usage information in the previous 6 dimensions, for a total of 10 dimensions, denoted as D2. After our preprocessing, the scale of D1 is 930×6 and D2 is 1152×10 , which is an appropriate data scale for the edge training model. The runtime of the task changes from about 10 s to 350 s, the chip usage-related dimensions change from 0 to 100, the memory usage-related dimensions change from about 0 to 10^4, and the disk usage changes from about 10^4 to 10^5.

For these two datasets, there are three points to note. First, the selected features are closely related to the runtime of the task. The more tasks run on the device, the fewer remaining resources, and the longer the task runtime. However, since the data only records resource information before the task runs, and resource changes during the task run are not recorded, the datasets are noisy. Second, we divided datasets D1 and D2 into training and testing datasets in a 7:3 ratio. Third, the data range of each dimension has a large deviation, so normalization is used for preprocessing the dataset. All data is compressed into intervals of [0, 20] based on the maximum and minimum values of all

TABLE I
CONFIGURATION OF PSO FOR MODELS TRAINED WITH PSO

Model	w_{max}	w_{min}	c_1	c_2	epoch	particle
ANFISU	1.5	0.5	0.5	0.5	30	30
ESODNFS-DNN	1.5	1	0.5	0.5	30	30
ANFIS	1	1	0.5	0.5	30	50

TABLE II
DNN CONFIGURATION FOR MODELS WITH DNN

Model	lr	layers	epoch
ESODNFS-DNN	null	[d,5],[5,5],[5,5],[5,5],[5,s]	null
HPFDNN	0.003	[d,64],[64,128],[128,64],[64,1]	1000
DNN	0.003	[d,64],[64,128],[128,64],[64,1]	1000

values in the dimension, which is conducive to adjusting the hyperparameters of the model.

B. Deployment of ESODNFS and Baseline Models

In this section, we introduce the parameter settings and deployment methods of ESODNFS, as well as some other models used in comparative experiments. As our model is used for prediction, we need to compare it with DNFS related to prediction. In addition, we have made improvements to the traditional T-S fuzzy model, so it is necessary to compare the models that have also been improved on the T-S fuzzy model. In addition, some traditional models also need to be used for comparison to provide more comprehensive proof of the performance of the model.

ESODNFS: In the experiment, we set s to 3, which means dividing the dataset into three parts and using three ANFISUs for training. Each ANFISU has the same configuration. In dataset D1, we set $a_{\max} = 6$, $a_{\min} = 3$, $K = 10$, and we set $a_{\max} = 5$, $a_{\min} = 2$, $K = 8$ in dataset D2. The configuration of the PSO is the same for both datasets, as shown in Table I. We trained these three models in parallel using a multiprocessing approach and finally used DNN for joint training. The configuration of DNN is shown in Table II, where lr indicates the learning rate. As DNN is trained using PSO, PSO configuration is also required, as shown in Table I.

HPFDNN [39]: HPFDNN was proposed for predicting cloud computing resource demand. In this model, the data is first passed through a fuzzy representation layer and a neural representation layer, fused, and then input into DNN for feature extraction and learning. The fuzzy representation layer uses Pythagorean fuzzy sets and Gaussian MFs, while the neural representation layer uses sigmoid functions. The DNN configuration of HPFDNN is shown in Table II. It should be noted that DNN also has dropout layers, where 10% of neurons are randomly discarded in each layer, and the momentum factor is set to 0.1. Furthermore, the linear descent learning rate is not utilized as its impact on the results is not significant.

DJFNN [41]: DJFNN is based on traditional T-S fuzzy models and has a structure similar to ANFIS. OR neurons have been proposed to represent more complex inference processes, and LSE has been used to solve the linear function coefficients. DJFNN uses a greedy algorithm to solve the weights between the MF layer and the OR layer, which are some 0/1 weights. The hyperparameters in DJFNN only have the number of MFs and rules for each dimension. In dataset D1, we set the number of MFs for all 5 input dimensions to 12 and the number of rules to 7. In dataset D2, the number of MFs is 9 and the number of rules is 6. This is the best configuration we have found for our problem.

ANFIS: In traditional ANFIS, the number of MFs for each dimension is set to be the same as the number of rules. We set the MF of ANFIS to Gaussian MF, and optimize the parameters using PSO. The configuration is shown in Table I. LSE is also used for calculating the coefficients of linear functions. The number of rules is set to 10.

DNN: The configuration of this DNN is the same as the DNN configuration in HPFDNN, but there is no setting for the momentum factor. Its configuration is shown in Table II.

C. Evaluation of ESODNFS Training Convergence Performance

In this section, we record the convergence performance of ESODNFS during training. When training ESODNFS, PSO is first used to train three ANFISUs in parallel, and then PSO is also used for joint training of DNN. Therefore, we can record the changes in the fitness functions, i.e. MSE, of the three ANFISUs and DNN during PSO operation to evaluate the convergence performance of the model training.

The convergence data during ESODNFS training is shown in Fig. 4. Whether in dataset D1 or D2, as well as ANFISU or DNN, PSO can significantly reduce MSE in the first few epochs and gradually flatten in the later stages, indicating the effectiveness of PSO in training. Horizontally, each ANFISU performs well on its dataset, while the training of DNN is on all training set samples, so it is normal for the final MSE of DNN to be higher than that of ANFISU. In addition, although 1st ANFISU(D2) has poor training performance, with an MSE of approximately 0.58, which is much higher than the other two ANFISUs, DNN training did not result in poor performance. Vertically, the initial MSE of ANFISU in the D1 dataset is approximately 0.3, while it exceeds 16 in the D2 dataset. However, in both datasets, the parameter initialization of PSO is the same. This means that the increase in dimensionality in the D2 dataset resulted in worse initial solutions, but the final convergence results were not significantly different between the two datasets, indicating that PSO is less likely to rely on the initial solution when optimizing parameters.

D. Evaluation of Joint Prediction Performance of ESODNFS's DNN

The DNN in ESODNFS weights the prediction results of multiple ANFISUs and obtains accurate results. We recorded the absolute value error $|t - \hat{t}|$ of a single sample prediction results of 3 ANFISUs and ESODNFS on the testing set, where $\hat{t}$ is the model's predicted value and t is the sample label. To better highlight the performance of ESODNFS, we sort the absolute value errors of ESODNFS output results in ascending order and remove a very small number of points with significant errors to better display the data.

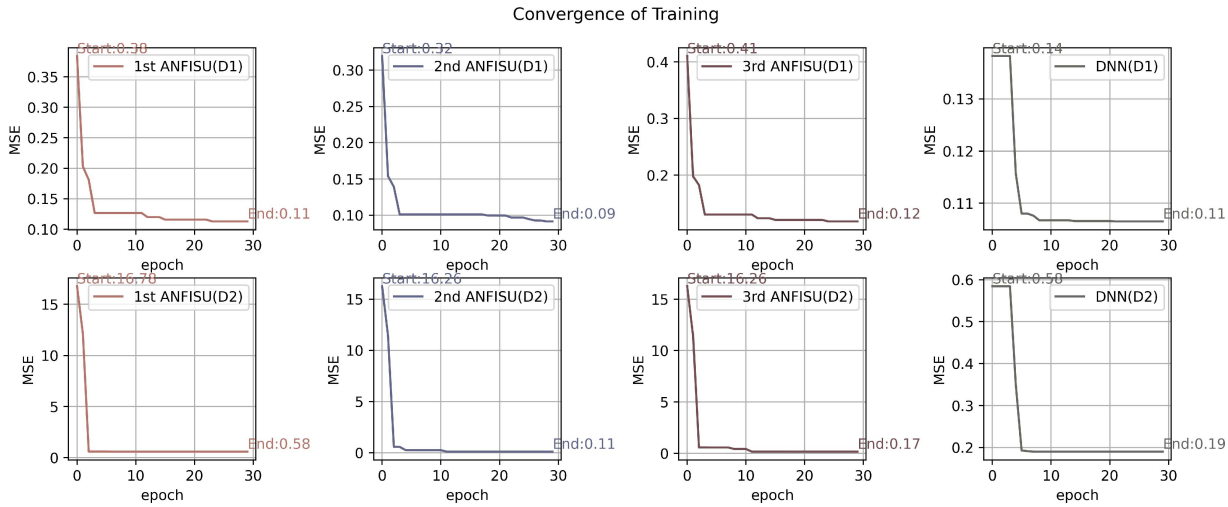

Fig. 4. During the ESODNFS training process, ANFISU and DNN use PSO for parameter optimization, achieving lower MSE.

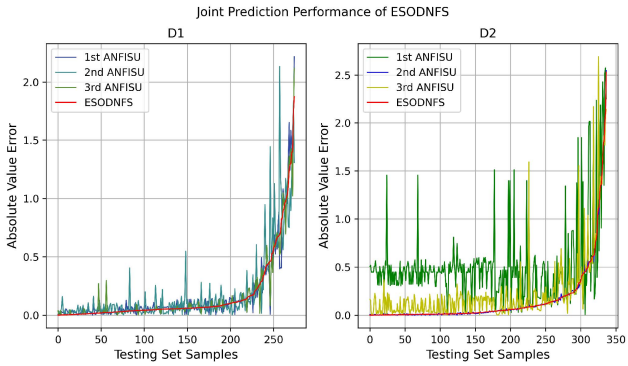

Fig. 5. The absolute value error of the predicted values of ANFISUs and ESODNFS on each sample in the testing set.

The absolute value error of the prediction results of ESODNFS and 3 ANFISUs for each sample on the testing dataset is shown in Fig. 5. On most samples, ESODNFS can fuse the output values of 3 ANFISUs to obtain reasonable prediction results, although the prediction error of ESODNFS is not always the best among the outputs of 3 AIFISUs. This is because each ANFISU learns different features, and for a certain sample on the testing set, some ANFISUs can accurately predict, while others have less than ideal prediction performance. On these ANFISUs that can be accurately tested, ESODNFS's DNN assigns them greater weights. However, ESODNFS inevitably needs to assign weights to other ANFISUs, resulting in ESODNFS not being optimal among all ANFISUs on certain samples.

In dataset D1, there is not much difference in the predictive performance of 3 ANFISUs. However, in some samples, the prediction value of one ANFISU has a large error. However, ESODNFS corrected the prediction result, thereby reducing the error. At the same time, it can be observed that over 80% of the samples have an absolute error of less than 0.25 in ESODNFS, while a small number of samples have poor prediction performance due to ANFISUs not having accurate prediction results on these samples.

In dataset D2, it is evident that the predictive performance of 1st ANFISU and 3 rd ANFISU is not as good, while 2nd ANFISU has a relatively good predictive performance. Therefore, ESODNFS's DNN assigns more weights to the 2nd ANFISU, resulting in ESODNFS's curve being very close to the 2nd ANFISU's curve. This indicates that ESODNFS has better robustness, even if the performance degradation caused by poor training performance of an ANFISU does not have a significant impact on the predictive performance of ESODNFS.

E. Evaluation of Model Prediction Accuracy and Training Time

In this section, we calculate the MSE of ESODNFS and the baseline models to compare the accuracy of the proposed model in prediction. To observe the accuracy of task runtime prediction more specifically, we converted the predicted values into the original time metric and calculated the average time error. Furthermore, we statistically analyzed the error distribution of the models on the testing set to gain a more specific understanding of the model's predictive performance. At the same time, the training time of the models is also counted to demonstrate that our ESODNFS can be trained more quickly.

As shown in Fig. 6, MSE on the training and testing sets of ESODNFS and other baseline models, as well as three ANFISUs, are recorded. From these data and the training process, we can summarize the following information: 1) Overall, ESODNFS has good predictive performance. Whether in the low-dimensional dataset D1 or the high-dimensional dataset D2, ESODNFS performs more accurately on the testing set compared to other models, with values of 0.5793 and 0.8161, respectively. In addition, the average time error of ESODNFS on the testing set is also smaller than most other baseline models, with values of 3.6292 s and 3.5673 s, respectively. 2) The results of ESODNFS on the training set are not optimal. However, some models have a small MSE on the training set, but a large MSE on the testing set. For example, ANFIS has an MSE of 0.135 and 2.054 on the training and testing sets in dataset D2, respectively.

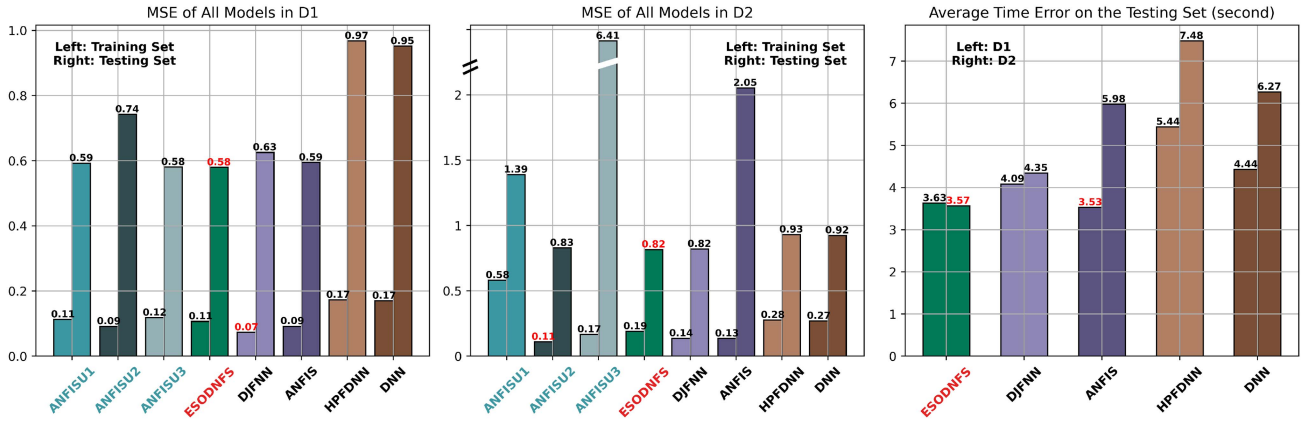

Fig. 6. The MSE of all models on the training and testing sets, as well as the average time error on the testing set.

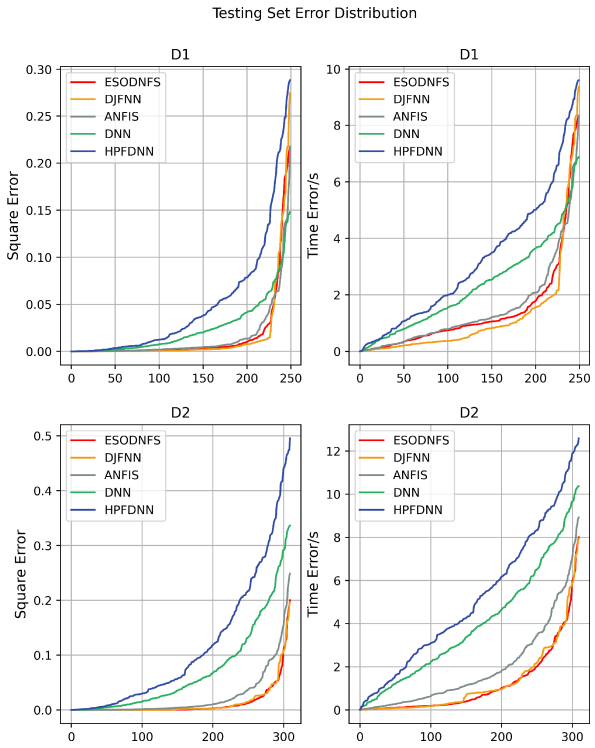

Fig. 7. The distribution of squared error and absolute time error on the testing set. The error of each model is calculated, and sorted, and the top 90% of the content is taken to better understand the error distribution.

This indicates that although ESODNFS is not as good as other models during training, it has a more prominent prediction effect in practical prediction due to its good generalization ability. 3) Due to the larger dimensions, larger data volume, and more noise in dataset D2, most models have a higher MSE in dataset D2 than in dataset D1. However, the MSE of DNN in dataset D2 is lower than that in D1, which may be related to the parameter initialization of the model. 4) The predictive performance of HPFDNN is not as good as DNN, which may be due to the incorrect parameters of the fuzzy representation layer during data fuzzification, resulting in the loss of features that can be extracted from the data.

TABLE III
TRAINING TIME FOR EACH MODEL

Model	Dataset D1		Dataset D2	
	Training Time(s)	MSE	Training Time(s)	MSE
ESODNFS	**360.88**	**0.5793**	**434.33**	**0.8161**
DJFNN	373.07	0.6255	632.61	0.8203
ANFIS	453.20	0.5948	674.62	2.0539
HPFDNN	453.67	0.9677	581.84	0.9308
DNN	441.06	0.9514	563.18	0.9243

The best performance in the comparison is highlighted in bold.

It is worth noting that some models have a larger MSE, but the average time error is smaller. This is because there is a significant error in predicting some samples, resulting in an amplification of the squared error. This indicates that MSE may be affected by prediction with small quantities but significant errors. Therefore, we statistically analyzed the error distribution of the model on the testing set, where 90% of the sample sizes with small errors are recorded, as this indicates the accuracy of the model prediction in most cases.

As shown in Fig. 7, the error of ESODNFS and DJFNN is small on most samples. In some cases (D1), the curve of DJFNN is below ESODNFS, indicating that more samples are predicted more accurately by DJFNN. However, ESODNFS has the smallest MSE, indicating that it can better handle extreme samples. From this graph, it can be observed that all three models based on the T-S fuzzy model have better predictive performance, indicating that the T-S fuzzy model is superior on noisy datasets.

We have calculated the time spent on training these models. As shown in Table III, due to the small number of parameters for ESODNFS and the parallel training of multiple ANFISUs, the training time for ESODNFS is the shortest in datasets D1 and D2. The training time of ESODNFS is shortened in dataset D1 by 3.27%, 20.37%, 20.45%, and 18.18%, and in dataset D2 by 31.34%, 35.62%, 25.35%, and 22.88%, compared to DJFNN, ANFIS, HPFDNN, and DNN. This means that ESODNFS can complete model training in a faster time, so that when the data of the end devices served by the edge server changes significantly, the model can be retrained more quickly,

achieving more accurate task runtime prediction services. It is worth noting that the training of multiple ANFISUs is conducted using parallel training on the same edge server. If distributed training is performed on multiple edge servers, it may be possible to achieve faster training speed, which is the potential of our ESODNFS.

VI. Conclusion

This paper introduces ESODNFS, a fast-training model with strong predictive and generalization capabilities, designed to adapt to rapid change of end device information in the Big Data era. We enhance ANFIS to create ANFISU, using PSO and LSE for efficient training. To further speed up training, we parallelize multiple ANFISUs on a partitioned dataset and use a DNN to weigh their outputs, improving prediction accuracy, model robustness, and training speed. Experiments show that ESODNFS, integrating parallel-trained ANFISUs with the task execution data from actual end devices, outperforms other models in prediction, training speed, and robustness.

However, as the T-S fuzzy model is further studied, more structures such as OR neurons are proposed, which may further improve the performance of our proposed ANFISU model. In addition, the interpretability of ESODNFS and how to enhance model security based on interpretability are also important research topics that require our consideration. Therefore, we will conduct further research on these issues in the future.

References

[1] S. U. Amin and M. S. Hossain, "Edge intelligence and Internet of Things in healthcare: A survey," *IEEE Access*, vol. 9, pp. 45–59, 2021.

[2] Y. Xiao, G. Shi, Y. Li, W. Saad, and H. V. Poor, "Toward self-learning edge intelligence in 6G," *IEEE Commun. Mag.*, vol. 58, no. 12, pp. 34–40, Dec. 2020.

[3] R. Gupta, D. Reebadiya, S. Tanwar, N. Kumar, and M. Guizani, "When blockchain meets edge intelligence: Trusted and security solutions for consumers," *IEEE Netw.*, vol. 35, no. 5, pp. 272–278, Sep./Oct. 2021.

[4] M. Gibbs and E. Kanjo, "Realising the power of edge intelligence: Addressing the challenges in AI and tinyML applications for edge computing," in *Proc. 2023 IEEE Int. Conf. Edge Comput. Commun.*, 2023, pp. 337–343.

[5] C.-J. Wu et al., "Machine learning at facebook: Understanding inference at the edge," in *Proc. 2019 IEEE Int. Symp. High Perform. Comput. Archit.*, 2019, pp. 331–344.

[6] B. Lu, J. Yang, W. Jiang, Y. Shi, and S. Ren, "One proxy device is enough for hardware-aware neural architecture search," in *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 3, pp. 1–34, 2021.

[7] C. Chandrashekar, P. Krishnadoss, V. Kedalu Poornachary, B. Ananthakrishnan, and K. Rangasamy, "HWACOA scheduler: Hybrid weighted ant colony optimization algorithm for task scheduling in cloud computing," *Appl. Sci.*, vol. 13, no. 6, 2023, Art. no. 3433.

[8] X. Wang, H. Lou, Z. Dong, C. Yu, and R. Lu, "Decomposition-based multi-objective evolutionary algorithm for virtual machine and task joint scheduling of cloud computing in data space," *Swarm Evol. Comput.*, vol. 77, 2023, Art. no. 101230.

[9] G. Chen, J. Qi, Y. Sun, X. Hu, Z. Dong, and Y. Sun, "A collaborative scheduling method for cloud computing heterogeneous workflows based on deep reinforcement learning," *Future Gener. Comput. Syst.*, vol. 141, pp. 284–297, 2023.

[10] R. Sing et al., "EMCS: An energy-efficient makespan cost-aware scheduling algorithm using evolutionary learning approach for cloud-fog-based IoT applications," *Sustainability*, vol. 14, no. 22, 2022, Art. no. 15096.

[11] X. Zhu and Y. Xiao, "Adaptive offloading and scheduling algorithm for Big Data based mobile edge computing," *Neurocomputing*, vol. 485, pp. 285–296, 2022.

[12] L. Zhou et al., "PREP: Predicting job runtime with job running path on supercomputers," in *Proc. 50th Int. Conf. Parallel Process.*, 2021, pp. 1–10.

[13] N. Talpur, S. J. Abdulkadir, H. Alhussian, M. H. Hasan, N. Aziz, and A. Bamhdi, "Deep neuro-fuzzy system application trends, challenges, and future perspectives: A systematic survey," *Artif. Intell. Rev.*, vol. 56, no. 2, pp. 865–913, 2023.

[14] D. A. Tedjopurnomo, Z. Bao, B. Zheng, F. M. Choudhury, and A. K. Qin, "A survey on modern deep neural network for traffic prediction: Trends, methods and challenges," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 4, pp. 1544–1561, Apr. 2022.

[15] P. Malhotra et al., "Deep neural networks for medical image segmentation," *J. Healthcare Eng.*, vol. 2022, 2022, Art. no. 9580991.

[16] M. Billah, X. Wang, J. Yu, and Y. Jiang, "Real-time goat face recognition using convolutional neural network," *Comput. Electron. Agriculture*, vol. 194, 2022, Art. no. 106730.

[17] N. Talpur et al., "A comprehensive review of deep neuro-fuzzy system architectures and their optimization methods," *Neural Comput. Appl.*, vol. 34, pp. 1837–1875, 2022.

[18] F. Zhang, S. P. Chowdhury, and M. Christakis, "DeepSearch: A simple and effective blackbox attack for deep neural networks," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2020, pp. 800–812.

[19] J. Serrano-Guerrero, F. P. Romero, and J. A. Olivas, "Fuzzy logic applied to opinion mining: A review," *Knowl.-Based Syst.*, vol. 222, 2021, Art. no. 107018.

[20] E. van Krieken, E. Acar, and F. van Harmelen, "Analyzing differentiable fuzzy logic operators," *Artif. Intell.*, vol. 302, 2022, Art. no. 103602.

[21] G. T. Reddy, M. P. Kumar Reddy, K. Lakshmanna, D. S. Rajput, R. Kaluri, and G. Srivastava, "Hybrid genetic algorithm and a fuzzy logic classifier for heart disease diagnosis," *Evol. Intell.*, vol. 13, pp. 185–196, 2020.

[22] J.-S. R. Jang, "ANFIS: Adaptive-network-based fuzzy inference system," *IEEE Tran. Syst., Man, Cybern.*, vol. 23, no. 3, pp. 665–685, May/Jun. 1993.

[23] R. Tabbussum and A. Qayoom Dar, "Performance evaluation of artificial intelligence paradigms—artificial neural networks, fuzzy logic, and adaptive neuro-fuzzy inference system for flood prediction," *Environ. Sci. Pollut. Res.*, vol. 28, no. 20, pp. 25265–25282, 2021.

[24] S. Pirmoradi, M. Teshnehlab, N. Zarghami, and A. Sharifi, "A self-organizing deep neuro-fuzzy system approach for classification of kidney cancer subtypes using mirna genomics data," *Comput. Methods Programs Biomed.*, vol. 206, 2021, Art. no. 106132.

[25] S. Bhanja, S. Metia, and A. Das, "A hybrid neuro-fuzzy prediction system with butterfly optimization algorithm for pm2. 5 forecasting," *Microsyst. Technol.*, vol. 28, no. 12, pp. 2577–2592, 2022.

[26] D. Sharma, G. S. Aujla, and R. Bajaj, "Deep neuro-fuzzy approach for risk and severity prediction using recommendation systems in connected health care," *Trans. Emerg. Telecommun. Technol.*, vol. 32, no. 7, 2021, Art. no. e4159.

[27] H. Al-Sayeh, S. Hagedorn, and K.-U. Sattler, "A gray-box modeling methodology for runtime prediction of apache spark jobs," *Distrib. Parallel Databases*, vol. 38, pp. 819–839, 2020.

[28] F. Chen, "Job runtime prediction of HPC cluster based on PC-transformer," *J. Supercomput.*, vol. 79, pp. 20208–20234, 2023.

[29] Q. Wang, J. Li, S. Wang, and G. Wu, "A novel two-step job runtime estimation method based on input parameters in HPC system," in *Proc. IEEE 4th Int. Conf. Cloud Comput. Big Data Anal.*, 2019, pp. 311–316.

[30] S. Ramachandran, M. L. Jayalal, M. Vasudevan, S. Das, and R. Jehadeesan, "Combining machine learning techniques and genetic algorithm for predicting run times of high performance computing jobs," *Appl. Soft Comput.*, vol. 165, 2024, Art. no. 112053.

[31] S. Zrigui, R. Y. de Camargo, A. Legrand, and D. Trystram, "Improving the performance of batch schedulers using online job runtime classification," *J. Parallel Distrib. Comput.*, vol. 164, pp. 83–95, 2022.

[32] W. Yang, X. Liao, D. Dong, and J. Yu, "Exploring job running path to predict runtime on multiple production supercomputers," *J. Parallel Distrib. Comput.*, vol. 175, pp. 109–120, 2023.

[33] F. Fanian and M. K. Rafsanjani, "Three-stage fuzzy-metaheuristic algorithm for smart cities: Scheduling mobile charging and automatic rule tuning in WRSNs," *Appl. Soft Comput.*, vol. 145, 2023, Art. no. 110599.

[34] S. Memarian, N. Behmanesh-Fard, P. Aryai, M. Shokouhifar, S. Mirjalili, and M. del Carmen Romero-Ternero, "TSFIS-GWO: Metaheuristic-driven takagi-sugeno fuzzy system adaptive real-time routing wbans," *Appl. Soft Comput.*, vol. 155, 2024, Art. no. 111427.

[35] M. Rodriguez, D. Arcos-Aviles, and W. Martinez, "Fuzzy logic-based energy management for isolated microgrid using meta-heuristic optimization algorithms," *Appl. Energy*, vol. 335, 2023, Art. no. 120771.
[36] Q. Tu, Q. Zhang, Z. Zhang, D. Gong, and M. Tang, "A deep spatiotemporal fuzzy neural network for subway passenger flow prediction with COVID-19 search engine data," *IEEE Trans. Fuzzy Syst.*, vol. 31, no. 2, pp. 394–406, Feb. 2023.
[37] Z. Sun, A. Liu, N. N. Xiong, S. Zhang, and T. Wang, "DNF-BLPP: An effective deep neuro-fuzzy based bilateral location privacy-preserving scheme for service in spatiotemporal crowdsourcing," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2024.3451236.
[38] L. Wang, Y. Qian, H. Xie, Y. Ding, and F. Guo, "Structured sparse regularization-based deep fuzzy networks for RNA N6-methyladenosine sites prediction," *IEEE Trans. Fuzzy Syst.*, to be published, doi: 10.1109/TFUZZ.2024.3428402.
[39] D. Chen, X. Zhang, L. Wang, and Z. Han, "Prediction of cloud resources demand based on hierarchical pythagorean fuzzy deep neural network," *IEEE Trans. Serv. Comput.*, vol. 14, no. 6, pp. 1890–1901, Nov./Dec. 2021.
[40] M. Kaveh and M. S. Mesgari, "Application of meta-heuristic algorithms for training neural networks and deep learning architectures: A comprehensive review," *Neural Process. Lett.*, vol. 55, no. 4, pp. 4519–4622, 2023.
[41] N. Wang, W. Pedrycz, W. Yao, X. Chen, and Y. Zhao, "Disjunctive fuzzy neural networks: A new splitting-based approach to designing a T–S fuzzy model," *IEEE Trans. Fuzzy Syst.*, vol. 30, no. 2, pp. 370–381, Feb. 2022.

Haijie Wu is currently working toward the MS degree with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China supervised by Dr. Weiwei Lin. His research interests mainly include cloud edge collaboration, fuzzy systems, edge computing, and AI algorithms.

Weiwei Lin (Senior Member, IEEE) received the BS and MS degrees from Nanchang University in 2001 and 2004, respectively, and the PhD degree in computer application from the South China University of Technology in 2007. Currently, he is a professor with the School of Computer Science and Engineering, South China University of Technology. His research interests include distributed systems, cloud computing, and AI application technologies. He is a distinguished member of CCF.

Wangbo Shen received the BS degree from Changsha University, Changsha, China in 2012 and 2016, and the MS degree from Central South University, Changsha, China in 2016 and 2019 respectively. Currently, he is working toward the PhD degree with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China supervised by Dr. Weiwei Lin. His research interests mainly include Kernel learning, AutoML and edge computing.

Xiumin Wang received the BS degree from the Department of Computer Science, Anhui Normal University, China, in 2006 and the PhD degree from the Department of Computer Science, University of Science and Technology of China, Hefei, China, and the Department of Computer Science, City University of Hong Kong, under a joint PhD program. She is now an associate professor with the School of Computer Science and Engineering, South China University of Technology, China. Her research interests include mobile computing, algorithm design and optimization.

C. L. Philip Chen (Fellow, IEEE) received the MS degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 1985, and the PhD degree in electrical engineering from Purdue University, West Lafayette, IN, USA, in 1988. He is currently the dean of the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His current research interests include systems, cybernetics, and computational intelligence. He is a fellow of AAAS and IAPR.

Keqin Li (Fellow, IEEE) is a SUNY distinguished professor of Computer Science with the State University of New York. He is also a National distinguished professor with Hunan University, China. His current research interests include, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, Big Data computing, high performance computing, computer architectures and systems, computer networking, ML, intelligent and soft computing. He is currently an associate editor of the ACM Computing Surveys and the CCF Transactions on High Performance Computing. He is an AAIA fellow. He is also a member of Academia Europaea (Academician of the Academy of Europe).